Research-Driven Curriculum Design

A Final Report Submitted in Completion of the Requirements of CPSC 490

Matthew Cline
Advised by Jay Lim

1. Introduction and Background

Computer science is the fastest-growing field of study in U.S. higher education. At Yale, the major has grown by 814% over the past 15 years – from 14 computer science graduates in the class of 2007 to 114 in the class of 2021 (Daniswara, Yale University Department of Computer Science). Interest in computer science among non-majors is even more striking. In September 2022, the Yale Daily News reported that "Computer science is the second-most popular major in Yale College, and total enrollment in computer science courses reached a new record high of 3,260 students for the fall 2022 semester, according to Department Chair Zhong Shao." (Stasiuk). This has coincided with new course offerings in interdisciplinary applications of computer science, such as "Python Programming for Humanities and Social Sciences," "Biomedical Software Design," and the "YData" course series. Additionally, there has been a proliferation of joint majors that invite students to combine computer science coursework with their education in electrical engineering, psychology, mathematics, fine arts, and, most recently, linguistics (Ye and Dhar). These trends signal that there is greater demand than ever for computer science education, specifically for courses that cater to new entrants to the field who may have little or no prior computer science experience.

This phenomenon is personal to me. I entered Yale as a political science major and during my first two years here, I took nearly exclusively political science or political science-adjacent courses. It wasn't until the fall semester of junior year that I took my first ever coding class – CPSC 100, a.k.a. CS50. The course immediately won me over, and I began incessantly meeting with students and professors (most consequentially with my now-advisor, Jay Lim) to seek advice on completing the computer science major in just three semesters. This acceleration through the academic curriculum coincided with my search for a summer internship during spring 2022 (immediately after finishing CS50) and my search for full-time roles during summer and fall 2022. My late entrance into college computer science and my rapid journey through it have offered me a unique perspective on and a passion for introductory computer science education.

When I learned that Professor Lim was designing a new introductory course to serve as a CS50 successor and a potential alternative to CPSC 202 as a pathway into the advanced coursework of CPSC 223, 323, and 365, I was immediately enthusiastic, and I proposed working on the design of the course as my senior project. This paper is the result of my work to develop a research-driven introductory computer science curriculum, informed by the latest conclusions and best practices in the field of computer science education (CSE).

2. Introductory Computer Science: A Teleological Perspective

When I began work on this course, I thought back to my political philosophy coursework and Aristotle's concept of *telos*. The Greek word refers to the "end goal" or "final cause" of something (Shields). Aristotle argues that to make good decisions for a system, decision-makers must first consider the *telos* of the system (Ibid). I wanted to explicitly use this teleological framework when designing this curriculum. So, what is the end goal or final cause of an introductory computer science course? Answering this question is deceptively tricky. There are numerous reasonable answers that, given the time and resource constraints on any course, might conflict with one another. Such potential answers include:

1) To prepare students for the next course in the curriculum sequence

2) To prepare students for all the department's core courses

3) To impart knowledge of a few fundamental computer science concepts

4) To serve as a breadth-first survey of multitudinous computer science subfields and topics

5) To sell prospective students on the major

To answer this question, I focused my research on a meta-analysis of computer science education literature.

3. Best Practices: A Study of Computer Science Education

In my research, I focused on studies of introductory curriculum specifically. These publications included studies of curriculum development and redesign as well as static introductory curricula. Research subjects include students at research universities, technical schools, liberal arts colleges, women's colleges, and HBCUs. Questions I sought to answer were:

1) What course attributes lead to positive student experiences in introductory courses?

2) How do entering students perceive computer science?

3) What factors lead to student attrition from computer science during the early curriculum stages?

Here are my key takeaways:

1) Students perceive a disconnect between introductory and advanced computer science. (Biggers, Carter, Giannakos, Hewner, Kapoor, Kinnunen, Lundberg, McEwan)

2) Entering students view computer science as an asocial field
(Barker, Biggers, Carter, Giannakos, Hewner, Guzdial, McEwan)

3) Entering students imagine computer science to be narrowly focused on programming (Bellino, Biggers, Carter, Hewner, Guzdial, Kinnunen, McEwan)

4) Multi- and interdisciplinary applications are a source of tremendous student interest in computer science (Carter, Hewner, Guzdial, Kinnunen)

I'll now cover some of the key papers that drove my conclusions:

In his paper *Undergraduate Conceptions of the Field of Computer Science*, Rose-Hulman Institute of Technology Professor Michael Hewner studied introductory computer science students' views of the computer science field. These students attended a variety of universities, including Duke University, Georgia Tech, and Spelman College. He interviewed these students with questions such as, "Whether they considered a particular course they mentioned was completely CS, or a mix of CS and some other field" and "If they considered someone doing a particular job to be 'doing Computer Science'" (109). Hewner notes that alignment between student expectations and reality for their coursework is important to their perseverance through the material, so helping early students form accurate expectations of computer science is a worthwhile endeavor. The most common misalignment that Hewner observes in his study is, "Students knew few specifics about the contents of future courses" (111). Hewner elaborates, "Almost anytime I asked students to speculate about the content of a future course, students would explain they really did not know much about what the course entailed. This was true of required courses, elective courses they were looking forward to taking, or even courses they had signed up for in the next semester" (111). Hewner also notes that introductory students have a narrow conception of computer science as just programming – building web pages, mobile apps, etc. (110-113). Hewner notes that this narrow conception of computer science and the perceived disconnect between introductory and advanced curricula are associated with less enthusiasm for the field (112-113).

Professor Lori Carter investigates this same phenomenon in her paper, *Why Students with an Apparent Aptitude for Computer Science Don't Choose to Major in Computer Science*. She hypothesizes that "students, male or female, don't pursue education in computing fields because they either have no information or incorrect information about what the study of computing involves and what sorts of careers are available to computing professionals" (27). To test this hypothesis, Carter studied 836 entering college first-years about their perception of computer science. This study sought to answer four questions (28):

> "1. What kind of experience and information do High School students, on the verge of making a decision about a college major, have about the field of Computer Science?
>
> 2. What do High School students think Computer Science is?
>
> 3. What perceptions regarding the computing field do students have that would influence them for or against choosing the major?
>
> 4. Are these answers significantly different for males and females?"

Based on student responses, Carter concludes that "The vast majority [80%] of students had no concept of what a Computer Science major entails," that just 2% of entering students had what could be fairly considered a "good grasp" of computer science, and that the number one reason students chose not to pursue computer science was "lack of desire to sit in front of a computer all day" (29-30). The other most common student-reported reasons for not pursuing computer science are that they already decided on another major, prefer a more people-oriented major/occupation, or are averse to programming. It's notable that, among students who listed an aversion to programming as a strike against studying computer science, only 11% of them had programming experience (30). Carter concludes her study with prescriptions for introductory computer science curricula that address the attitudes she uncovered (30-31):

1) Offer multidisciplinary and cross-disciplinary programs. Carter writes, “The number one reason for females and the number three reason for males that might influence them towards a major in Computer Science is the desire to use computing in another field. Students need to see that Computer Scientists were involved in mapping the human genome (bioinformatics), hurricane prediction (environmental modeling), and creating a machine to perform Lasik surgery (medicine).”

2) Fix computing science's image. Carter writes, “Most students who gave a description of Computer Science saw it as programming or advanced computer use and were rejecting it because they did not desire to sit in front of a computer all day. The educated student would learn that many aspects of Computer Science require significant people interaction. Along with those listed in the previous paragraph, computers are used for special effects in movies, to improve the quality of life for people with missing limbs, and for allowing communication for people with speech impediments.”

3) Make CS courses fun. Carter writes, “The men’s top reason for choosing CS a major was interest in Computer Games. However, the women’s top reason was to use CS in another field. Consequently, this goal should be restated to say that it is important to make CS courses creative and relevant.”

In their paper *Student Perceptions of Computer Science: A Retention Study Comparing Graduating Seniors vs. CS Leavers*, Georgia Tech Professors Maureen Briggs, Anne Brauer, and Tuba Yilmaz perform a comparative analysis of students who graduated from the computer science major at Georgia Institute of Technology (the Stayers) versus those who chose to leave the major (the Leavers). They identify multiple axes along which the Stayer and Leaver experiences differed:

1) Level of preparation. 65% of Stayers report adequate levels of preparation for university computer science before university, while just 38% of Leavers report the same (403).

2) "Real-world" conception of CS: Stayers report a more expansive conception of computer science focused on solving problems, whereas Leavers' view of computer science is more restricted to the study of software, computers, etc. (404).

3) Code-only conception of CS: Stayers specified that CS is not "about information technology (IT), or about fixing hardware, or about purely being a 'code monkey.'" However, "precisely these types of definitions appeared significantly often in the responses of the Leavers" (404).

The authors conclude that "Students who left the CS major have an overwhelming perception that CS is an asocial, coding-only field with little connection to the outside world" (405). They add, "For the Leavers, exposure to computer science was often limited to the intensely programming-focused first-year courses. Students described these courses as full of assignments in which they could not see real world relevance" (405-406). The authors write that the "loss of interest" reported by Leavers is explained by "the undesirability of a career of only coding. This indicates that these students likely develop a limited view of what their career options as computer science majors actually are" (406).

4. An Uneasy Alliance: Bridging the Gap between Academia and Industry

Another subject of my computer science education research was academia-industry alignment. Research of computer science curricula finds that solving real-world problems is a powerful driver of students' enthusiasm for computer science (Barker, Bellino, Giannakos, Hewner and Guzdial, Kapoor and Gardner-McCune, Lundberg and Krogstie, Vega). This indicates that some level of alignment with industry is desirable for introductory computer science courses specifically.

My research in this field of CSE yielded a few key conclusions.

> **1. Soft skills, specifically technical communication, are consistently recognized by industry professionals as skills of foremost importance for new engineers** (Gurcan and Kose, Joint Task Force 2013, Joint Task Force 2014, Jun, O'Leary)
>
> **2. Project-based learning most closely models industry workflows and allows students to develop their skills of independent learning and design decision making** (Gurcan and Kose, Joint Task Force 2013, Joint Task Force 2014, Jun, O'Leary, Vega)
>
> **3. Just-in-Time curriculum is a powerful tool to synthesize project-based learning with theoretical concepts** (Phillips)`

One of my most worthwhile discoveries was of the *Software Engineering Education Knowledge (SEEK)* guidelines, developed by working groups at the Institute of Electrical and Electronics Engineers (IEEE) and the Association for Computing Machinery (ACM). Notably, the SEEK guidelines outline ten key "knowledge units" that comprise an industry preparatory computer science curriculum (Joint Task Force 2013, Joint Task Force 2014). One such unit is "Professional Practice," which includes "the knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner. The study of professional practices includes the areas of technical communication, group dynamics and psychology, and social and professional responsibilities" (Joint Task Force 2013).

The importance of this skillset is also reflected in O'Leary et. al.'s paper, *Developing a Software Engineering Curriculum for the Emerging Software Industry in China*. In this paper, the authors chronicle their involvement in the Emersion project, which sought to develop curriculum standards for Chinese technical universities. As part of this development process, the authors surveyed technical industry professionals to investigate the required skills for new engineers. From this survey, the authors learned that "There was no substantial agreement among respondents on the actual technologies or methodologies. However, the vast majority of respondents were of the opinion that a firm grounding in the fundamentals of computing and software engineering was of much greater benefit than specific technologies, programming languages or platforms" (4). From their research, the authors propose four "pillars" of computer science education (4). One such pillar relates to education components specific to China (language and civics courses, etc.). They deem the other three generalizable. One pillar, *Industrial Involvement*, entails the integration of industry placement into university computer science departments (similar to the University of Waterloo's co-op program) (5). This is beyond the scope of this project. Another pillar, *Key Transferable Skills*, refers to communication, presentation, and collaboration skills (4). The fourth pillar is *Key Technologies* (5). As noted previously, there is no widespread industry consensus that any single technology is essential, so instead, the authors argue that a curriculum's key technologies should be chosen based on their effectiveness as learning tools, i.e. how well they facilitate student comprehension of programming fundamentals (5). The authors propose a four-year curriculum framework, in which year one's focus is "to provide students with foundational programming and problem solving skills, good communication and presentation skills as well as a deep appreciation of the main topics in computing" (6). This emphasis on communication is also supported by researchers Gurcan and Rose in their paper *Analysis of Software Engineering Industry Needs and Trends: Implications for Education*. In it, they implement LDA-based topic modeling of computer science-related job listings and find "Communication Skill" to be the highest-rated keyword – appearing in 15.9% of job listings, even more often than degree requirements at 11% (1365).

Another strategy to more closely align introductory coursework with industry is to implement project-based learning. The purpose is twofold. First, it provides a natural setting for collaboration. Huang Jun of China Jiliang University writes that the practice "can greatly help computer undergraduates better understand the significance of social aspects and improve their teamwork skills and innovation… Results from our first 3 years indicate the methodology is manageable and effective for increasing students' teamwork capability…" (652). An IEEE/ACM working group similarly concludes, "In general, students can best learn to apply much of the material defined in the Software Engineering KA [knowledge area] by participating in a project" (Join Task Force 174). They add that "While organizing and running effective projects within the academic framework can be challenging, the best way to learn to apply software engineering theory and knowledge is in the practical environment of a project" and conclude that, "there is increasing evidence that students learn to apply software engineering principles more effectively through an iterative approach, where students have the opportunity to work through a development cycle, assess their work, and then apply the knowledge gained through their assessment to another development cycle" (174).

A different working group composed of university faculty and Microsoft and Google engineers published a guide to implementing a cloud computing curriculum at the university level – *Cloud Computing: Developing Contemporary Computer Science Curriculum for a Cloud-First Future*. A key insight from this paper is its endorsement of project-based learning, as it requires students to independently learn new technologies (reading documentation, debugging new errors, etc.). They write, “In fact, cloud computing is fundamentally suited to incorporating active learning exercises into curriculum design, an approach that is well received by students…” (136). They also propose capture-the-flag “hacking” exercises as cybersecurity pedagogy (144).

A pedagogical approach that meshes well with project-based learning is just-in-time (JIT) curriculum design. JIT refers to teaching concepts as they are required to complete assigned tasks. Within this paradigm, the introduction of new topics coincides with those topics’ immediate relevance. Phillips, Stevenson, and Wick, all professors of computer science at the University of Wisconsin, outline this approach in their paper *Implementing CC2001: A Breadth-First Introductory Course for a Just-in-Time Curriculum Design*. They describe the approach: “The JIT model calls for the introduction of units of knowledge just prior to when they are needed in support of discussions of both fundamental principles and new technologies, and even then only at the level necessary to handle those topics. For example, we introduce database queries using SQL in our first course just in time for the students to build a series of HTML forms and JavaServer Pages to provide a limited form of search capability to a small sample database” (239). The authors identify three advantages to the JIT model (240).

> “1) Since topics covered in early courses are directly expanded (with little or no review) in more advanced courses, students are forced to develop better retention skills.
>
> 2) Since students are presented with material from the same core units of knowledge multiple times over the entire curriculum, each time with increasing levels of

sophistication and depth, students are better prepared to understand the topics when covered in detail, and to systematically reflect on those topics as their overall knowledge of the discipline grows.

3) Students are directly motivated to study the theory of core units as that theory is immediately applied at the level learned to other topics within the computer science curriculum."

5. Q&A: Synthesizing Research Findings with Research Questions

After researching computer science education best practices and industry recommendations, I returned to my initial question – "What is the end goal or final cause of an introductory computer science course?" Armed with the results of my research, I arrived at three primary goals for Yale's new introductory course. The course will aim to provide students with:

1) As full and accurate a picture as possible of the diverse applications of computer science to real-world problems

2) The theoretical and technical foundation to succeed in the Yale computer science major's core courses

3) A clear understanding of what further academic study of computer science looks like

I also designated two secondary goals for the course, to be pursued contingent on compatibility with the primary goals. The course will also aim to provide students with:

1) Opportunities to complete software projects that are relevant to real-world problems

2) Opportunities to make high-level design decisions and to explain those decisions to others

I then synthesized explicit answers to my research questions:

1) What course attributes lead to positive student outcomes in introductory courses?

Answer: The best introductory computer science courses are:

Situated: There's a clear connection between introductory coursework and the rest of the department

Interdisciplinary: The course appeals to students of diverse interests and backgrounds by showcasing the inter- and multidisciplinary applications of computer science

2) How do entering students perceive computer science?

Entering students perceive computer science as a largely **asocial** field of study, and they generally perceive it as **narrowly focused on programming**.

3) What factors lead to student attrition from computer science at the early curriculum stages?

Students who exit the major at the early stages often report feeling a **lack of relevance** of their coursework to their interests or academic or career aspirations.

6. The Syllabus: Charting a Course in Computer Science

With the above findings in mind, I knew this new course must be situated, interdisciplinary, relevant, and social. At this point in the research process, I began to consider the logistics of the course. This new course, with the working title CPSC 101, would meet twice weekly with a weekly section. The coding languages for the course will be C and Python, as with CS50. Given the course's goal of showcasing diverse applications of computer science and connecting with more advanced courses, the first curriculum design choice is to structure CPSC 101 as a survey course. In this model, I segment the curriculum into six two-week modules, each with a unified content focus and associated deliverables.

When selecting the content theme for each module, the *situatedness* was of foremost importance. In other words, how does each module connect with students' future academic studies? Since all computer science majors must take CPSC 223, 323, and 365, CPSC 101 would do well to offer students a sense of the material and relevance of those courses. As such, the first three modules of the CPSC 101 syllabus are Data Structures, Algorithms, and Systems.

The Data Structures module covers the expected topics: a selection of data structures (arrays, linked lists, stacks, queues, heaps, hash maps), their associated operations (insert, delete, index) and the time complexities of each, and appropriate applications for each. This module will include code comparisons of C and Python code, but it will focus on Python. There are a few reasons for this. First, the goal of this module is for students to gain a strong grasp of the theory behind these structures and their primary usages; the focus is not on memory management and implementation specifics. Python is best suited to communicate these basics. Additionally, CPSC 223 gives a full treatment of data structures in C, which necessarily devotes substantial instruction and problem set hours to the specifics of the C programming language. I feel that students will have a better experience in 223 if they enter the course with a grasp of the data structures themselves. The deliverables for this and all modules are tentative. However, in the current version of the syllabus, this unit will culminate in students coding software caches with different structures and different eviction policies (least recently used, least frequently used, etc.). This assignment allows an opportunity to discuss caching and its utility at a high level, which will provide helpful context to students who eventually take 323. It is also a topic with versatile applications to software that students interact with often (Chrome browser history, Netflix video streaming, Twitter database query caching, etc.). This work of revealing the principles and abstractions that underpin software that students interact with regularly helps achieve the relevance that this course seeks to furnish. Finally, this module allows an abstraction/task to guide students' implementation. My earlier research findings indicate that "design a cache to store the 100 most frequently fetched Twitter queries" is a more compelling and exciting assignment than "code an LFU cache using a doubly-linked list and hash table." Furthermore, an assignment like this one allows students to begin making design tradeoffs between suboptimal and optimal solutions (such a problem could be solved with an array at a much higher time cost) and to consider data structures' appropriateness for different applications.

The Algorithms module will expand on the Data Structures module by deploying the data structure selection and complexity analysis in the context of formalized problem-solving procedures. This module will focus on the main topics of CPSC 365 – graphs and recursion. A particular focus of this module is the code implementation of these algorithmic principles, as CPSC 365 is exclusively a pencil and paper course. By the end of this module, students will be able to implement the most crucial pathfinding algorithms in Python code, reason about which data structures are appropriate for which algorithms and why, and grasp the principles of recursion in code (the call stack, how this relates to space complexity, the importance of a base case, memoization + hash map's relevance to it, tail-end recursion, the dangers of recursion in production software). For this section's module, students will develop a rudimentary version of a navigation app (Google Maps, Apple Maps, etc.) with a command line interface that supports multi-modal navigation through downtown New Haven. I drafted this problem set and will discuss it more thoroughly in subsequent sections.

The Systems Programming module will focus on the basics of virtual memory, address spaces, and assembly. Depending on whether this course becomes an alternative to CPSC 202, this module may require some refactoring to include Von Neumann architecture, binary, and logic gates. This module will naturally employ the C programming language. I've had the least involvement in the content organization of this module, as Professor Lim has ample materials from his instruction of Systems Programming and Compilers. However, the deliverable for this unit is one of the first that we decided on. CPSC 223 and 323 both require extensive use of debugging tools such as Valgrind and GDB; however, the instruction of these tools never receives full treatment from the courses. The expectation is that students will learn them on the fly through office hours and supplemental videos. Since these tools are essential to success in Yale computer science and since grasping them deepens one's understanding of control flow and memory management, the goal for this module's deliverable is to familiarize students with these tools. As such, the deliverable for this module will more closely resemble a traditional problem set than the others. In it, students will receive broken code and use these tools to remedy it. This presents an exciting opportunity for students to develop their skills at understanding and identifying bugs in a code base that others have written. This better reflects the nature of real-world software engineering work than the usual model of students writing the bulk of code for their problem sets themselves.

The completion of these three modules brings the course and its students to the midterm. By this point, they have tangible examples of what further computer science education looks like; they have a strong theoretical foundation in the basic concepts of computer science education, and they've connected these concepts with real-world applications that they interact with daily. The goal for the remaining modules is to give students a sense of the broader course offerings within Yale computer science and to deepen the connection between computer science education and real-world considerations. Based on course demand, our wish to incorporate low-level and high-level programming, and a desire to teach modules that students will find fun, I chose cybersecurity and artificial intelligence/machine learning as the subjects of the next two modules.

The cybersecurity module will focus on security vulnerabilities both near and far from the hardware. Adjacent topics include assembly, calling convention, memory allocation, web programming, networking, databases, encryption, and blockchain. The module is structured around two security vulnerabilities – SQL injection and buffer overflow. To teach the concepts underpinning each vulnerability, students will study famous examples of each (such as the Morris Worm and Code Red buffer overflow attacks and the Sony Pictures and Heartland Payment Systems SQL injection attacks). The deliverable for this module will be a series of capture-the-flag (CTF) attacks in which students will use their newfound knowledge of these attacks to exploit vulnerabilities in code to gain access to secure servers and manipulate web databases. The module will emphasize opportunities for further study in courses like CPSC 364, 413, 422, 433, 437, and 467.

The AI/ML module will cover topics such as statistical learning theory, classification and regression algorithms, neural networks, deep learning, and reinforcement learning. The module will also cover and differentiate between popular tools, technologies, and libraries such as Scikit-learn, TensorFlow, PyTorch, and Generative Pre-Trained Transformers 1-4. This module's deliverable will require students to develop a rudimentary version of tech startup OpenDoor's home appraisal algorithm. Using the Boston housing dataset available through Scikit-learn, students can build a model that can predict the selling price of a house based on its features such as the number of bedrooms, bathrooms, square footage, etc. Completion of this project entails loading and exploring the dataset, preprocessing the data, building and training a machine learning model to predict housing prices using a regression algorithm such as linear regression or decision trees, evaluating model performance using appropriate metrics such as mean squared error, root mean squared error, and R-squared, tuning the model, and visualizing the results. The module will elucidate the distinctions between Yale's computer science and statistics and data science AI/ML courses so that students can make an informed decision about whether and how further exploration of this topic is right for them.

Completion of the modules so far leaves three weeks of instruction time in the semester, plus reading period and finals period. Here, I plan a divergence of deliverables and lecture content. At this point, students will pivot their focus to their final projects in the course – a substantial coding project in a language of their choice that addresses a real-world problem and requires high-level design tradeoffs. The CS50 final project allows two weeks, and even in that compressed time window, the beginning programmers develop projects of impressive scope. With another semester of instruction, a broad foundation in computer science fundamentals, experience learning new technologies from reading documentation, and more time, the expectation is that students' final projects will meaningfully exceed the scope of the CS50 final projects. Starting with module six, developing these projects will be the students' only deliverable. Lectures will pivot to focus on the practical skills of personal project development (version control, defining a minimum viable product (MVP), prototyping and user testing, logging, automation, scripting, and technical communication) as well as open-ended q&a. Remaining lectures will be devoted to guest lectures, in which professors of upper-level elective courses visit the class to offer an insight into what to expect from their courses. Again, this serves the goal of clarifying what the study of computer science is and what students can expect from further study in the department. The course culminates in the submission of code for the final project as well as a presentation in which students explain their project, technologies, and design trade-offs for a non-technical audience.

7. Case Study: Developing the Algorithms Deliverable

As one deliverable for this project, I drafted the deliverable for the Algorithms module. Inspiration for the project came from a real question I received in a technical interview for a software engineer role with Amazon. The goals for this assignment are for students to:

1) Solidify their understanding of graphs

2) Become confident representing graphs in code

3) Gain familiarity with graph traversal

4) Learn to implement common graph traversal algorithms

5) Deepen their understanding of data structures and their appropriateness for different applications

6) Appreciate the tradeoffs between optimizing for different priorities (runtime, exhaustiveness, etc.)

7) Sharpen their overall coding abilities

The full Jupyter notebook with the project code is included with my final submission. For this section of the report, I'll elaborate on my rationale for composing the project as I did. I encourage anyone to access the project code to read alongside this report.

The problem set begins by defining terminology – graphs, nodes/vertices, edges, connected, acyclic, directed. It continues to expand on real-world applications of graphs to navigation and social networking, before going over ways to represent graphs as code (adjacency list) and to interact with that code (iterating over hash maps, iterating over key-value pairs when the value is itself an iterable object). It then focuses on graph traversal, beginning with imagining a linked list as a graph and having students traverse the list. Students then learn a simple graph exploration algorithm (as covered in Professor Wibisono's CPSC 365 course) and reason about prerequisites for its correctness (graph connectedness).

At this point, students learn trees, breadth-first search, and level-order traversal, and the problem set's focus pivots to the real-world example. Students are presented with a list of edges that comprise connections between destinations in downtown New Haven. Students will work with this graph representation for the rest of the problem set to implement and understand the operations of various graph traversal algorithms and the data structures used to realize them, including breadth-first search, iterative and recursive depth-first search, breadth-first search with priority, Dijkstra's with dynamic edge weighting, and A* with a straight-line distance heuristic function. Questions embedded throughout the project invite students to reason about the tradeoffs between different algorithms and different attributes of the underlying graph that might make one algorithm preferable to another. Students also use auxiliary data structures (arrays, hash sets, frozen sets) to realize these traversal algorithms, as well as variable scoping and path memory. The project culminates in the implementation of all these algorithms within a command line interface, which will provide a cursory overview of object-oriented programming.

This project typifies what I think great computer science assignments do. It is interactive, and it's paced with organic stopping points throughout; each section is bookended by objectives and knowledge checks, so students can be sure that they are retaining the essential information from each part of the assignment. I took tremendous inspiration from CS50 and the Odin Project (https://www.theodinproject.com/) in the structure of the assignment. It guides students through objectives but leaves implementation specifics up to them, and each piece builds on the next until, by the end of the project, students have derived a perhaps surprisingly robust application that reflects a real-world software solution for a real-world problem.

9. Going Forward: What's Next for CPSC 101

Getting this far in the project has only energized me more about this course. I truly believe that this course can change students' perceptions of computer science and raise Yale's already high stature as an institution of computer science pedagogy. Students will exit this course with a portfolio of robust software applications, the ability to trade off different data structures for different tasks, a coherent set of interests in different subfields of computer science, a sense of computer science theory's significance to systems they interact with daily, fluency in two of the most versatile programming languages, experience conveying technical decisions in layperson's language, and an informed opinion about whether further computer science study or career pursuit is right for them. This course is the culmination of the latest research and best practices from the computer science education field – it doesn't ask, "What do we want to teach?" but rather, "What do students need to know?" In short, it's a computer science course that I would have loved to have taken as I was finding my footing in this field. As the semester draws to a close, I've included my notes and code samples with this report so that others who work on developing this course can pick up where this project leaves off. With just the framework already in place, the course is on track to be situated, interdisciplinary, and relevant. The social/collaborative aspects of the course are worth the attention of those who work on it going forward. With that completed and once the infrastructure for the assignments is fully functional, this course will be ready for a trial run. My hopes for it are very high.

Bibliography

Albarakati, Noura, et al. "Rethinking CS0 to Improve Performance and Retention." ACE '21: Proceedings of the 23rd Australasian Computing Education Conference, February 2021, pp. 131-137, doi:10.1145/3441636.3442314.

Barker, Lecia J., Charlie McDowell, and Kimberly Kalahar. "Exploring factors that influence computer science introductory course students to persist in the major." SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education, March 2009, pp. 153-157. doi:10.1145/1508865.1508923.

Bellino, Alessio, et al. "A Real-world Approach to Motivate Students on the First Class of a Computer Science Course." ACM Transactions on Computing Education, vol. 21, no. 3, 2021, article 22, pp. 1-23, doi: 10.1145/3445982.

Bui, Giang, et al. "Differences in Intention to Major in Computing Across CS1." SIGCSE 2023: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2, March 2023, pp. 1326, doi: 10.1145/3545947.3576273

Biggers, Maureen, et al. "Student perceptions of computer science: a retention study comparing graduating seniors with cs leavers." ACM SIGCSE Bulletin, vol. 40, no. 1, 12 March 2008, pp. 402-406, doi: 10.1145/1352322.1352274.

Blanchard, Jeremiah J., et al. "How Perceptions of Programming Differ in Children with and without Prior Experience: (Abstract Only)." Proceedings of the 49th ACM Technical Symposium on Computer Science Education, ACM, Feb. 2018, p. 1099, doi: 10.1145/3159450.3162308.

Carter, Lori. "Why students with an apparent aptitude for computer science don't choose to major in computer science." ACM SIGCSE Bulletin, vol. 38, no. 1, 03 March 2006, pp. 27-31, doi: 10.1145/1124706.1121352.

Colomo-Palacios, Ricardo, Pedro Soto-Acosta, and Cristina Casado-Lumbreras. "A Vision on the Evolution of Perceptions of Professional Practice: The Case of IT." International Journal of Human Capital and Information Technology Professionals, vol. 6, no. 1, 2015, pp. 65-78.

"Co-op and Careers." University of Waterloo, n.d., https://uwaterloo.ca/future-students/co-op. Accessed 4 May 2023.

Cukierman, Diana. "Predicting Success in University First Year Computing Science Courses: The Role of Student Participation in Reflective Learning Activities and in I-clicker Activities." ITiCSE '15: Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, ACM, June 2015, pp. 248-253, doi: 10.1145/2729094.2742623.

Cukierman, Diana, et al. "Addressing challenges students face in first-year university Computing Science and Engineering Science courses: Overview of a needs assessment and workshop." Proceedings of the Western Canadian Conference on Computing Education, 2014, pp. 1-5, doi:10.1145/2597959.2597976.

Daniswara, John Amadeo. "Up & To The Right - The Growth of Computer Science at Yale." *Medium*, 18 Jan. 2018, https://medium.com/@johnamadeo/up-to-the-right-the-rise-of-computer-science-at-yale-fd9983b0596c.

Dorodchi, Mohsen, and Nasrin Dehbozorgi. "Addressing the Paradox of Fun and Rigor in Learning Programming." Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, ACM, June 2017, pp. 370, doi: 10.1145/3059009.3073004.

Duran, Rodrigo, et al. "Retention in 2017--18 Higher Education Computing Programs in the United States." ACM Inroads, vol. 12, no. 2, May 2021, pp. 18-28, doi:10.1145/3448356.

Egan, Rylan, et al. "The academic enhancement program in introductory CS: a workshop framework description and evaluation." Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, June 2011, pp. 278-282. doi: 10.1145/1999747.1999825.

Foster, Derek, et al. "Cloud computing: developing contemporary computer science curriculum for a cloud-first future." ITiCSE 2018 Companion: Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, July 2018, pp. 130-147, doi: 10.1145/3293881.3295781.

Giannakos, Michail N., et al. "Understanding student retention in computer science education: The role of environment, gains, barriers and usefulness." Education and Information Technologies, vol. 22, no. 6, 2017, pp. 2365-2382. doi:10.1007/s10639-016-9538-1.

Gurcan, Fatih and Cemal Kose. “Analysis of software engineering industry needs and trends: implications for education.” *International Journal of Engineering Education* 33 (2017): 1361-1368.

Hewner, Michael. "Undergraduate conceptions of the field of computer science." ICER '13: Proceedings of the ninth annual international ACM conference on International computing education research, August 2013, pp. 107-114, doi: 10.1145/2493394.2493414.

Hewner, Michael, and Maria Knobelsdorf. "Understanding computing stereotypes with self-categorization theory." Koli '08: Proceedings of the 8th International Conference on Computing Education Research, November 2008, pp. 72-75, doi:10.1145/1595356.1595368.

Hewner, Michael and Mark Guzdial. "Attitudes about computing in postsecondary graduates." Proceedings of the Fourth international Workshop on Computing Education Research, September 2008, pp. 71-78, doi: 10.1145/1404520.1404528.

Hewner, Michael, and Mark Guzdial. "How CS majors select a specialization." Proceedings of the seventh international workshop on Computing education research, August 2011, pp. 11-18, doi: 10.1145/2016911.2016916.

Isenegger, Kathleen, et al. "Goal-Congruity Theory Predicts Students' Sense of Belonging in Computing Across Racial/Ethnic Groups." Proceedings of the 54th ACM Technical Symposium on Computer Science Education, vol. 1, March 2023, pp. 1069-1075, doi: 10.1145/3545945.3569834.

Isenegger, Kathleen, et al. "Understanding and Expanding College Students' Perceptions of Computing's Social Impact." 2021 Conference on Research in Equitable and Sustained Participation in Engineering, Computing, and Technology (RESPECT), 23-27 May 2021, doi: 10.1109/RESPECT51740.2021.9620589.

Joint Task Force on Computing Curricula, Association for Computing Machinery, & IEEE Computer Society. (2013). Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. Association for Computing Machinery. doi:10.1145/2534860

Joint Task Force on Computing Curricula, IEEE Computer Society, and Association for Computing Machinery. "SE 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering." Computer, vol. 48, no. 11, Nov. 2015, pp. 106-109. IEEE, doi: 10.1109/MC.2015.345.

Jun, Huang. "Improving undergraduates' teamwork skills by adapting project-based learning methodology." IEEE, 2010, doi: 10.1109/ICCSE.2010.5593527.

Kapoor, Amanpreet, and Christina Gardner-McCune. "Considerations for switching: exploring factors behind CS students' desire to leave a CS major." Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, ACM, July 2018, pp. 290-295, doi: 10.1145/3197091.3197113.

Kapoor, Amanpreet, and Christina Gardner-McCune. "Understanding Professional Identities and Goals of Computer Science Undergraduate Students." Proceedings of the 49th ACM Technical Symposium on Computer Science Education, ACM, Feb. 2018, pp. 191-196, doi: 10.1145/3159450.3159474.

Kinnunen, Päivi et al. "Understanding initial undergraduate expectations and identity in computing studies." European Journal of Engineering Education, vol. 43, no. 2, 2018, pp. 201-218. doi: 10.1080/03043797.2016.1146233.

Lewis, Colleen, et al. "Alignment of Goals and Perceptions of Computing Predicts Students' Sense of Belonging in Computing." ICER '19: Proceedings of the 2019 ACM Conference on International Computing Education Research, July 2019, pp. 11-19, doi:10.1145/3291279.3339426.

Liargkovas, Georgios, et al. "Software Engineering Education Knowledge Versus Industrial Needs." IEEE Transactions on Education, vol. 65, no. 3, August 2022, pp. 419-427, doi: 10.1109/TE.2021.3123889.

Lowe, Tony. "Rethinking assessment in early computing courses." 2022 IEEE Frontiers in Education Conference (FIE), 8-11 October 2022, doi: 10.1109/FIE56618.2022.9962573.

McEwan, Tom and Aysu McConnell. "Young people's perceptions of computing careers." 2013 IEEE Frontiers in Education Conference (FIE), 2013, pp. 1597-1603, doi: 10.1109/FIE.2013.6685108.

Lombart, Cécile et al. "Tips and Tricks for Changing the Way Young People Conceive Computer Science." Informatics in Schools: Engaging Learners in Computational Thinking, edited by Koli, Kerttu and Laanpere, Mart, vol. 12518, Springer, Cham, 2020, pp. 89-100. Lecture Notes in Computer Science, doi: 10.1007/978-3-030-63212-0_7.

Lundberg, Gunhild M., and Birgit R. Krogstie. "Employability Through Imagination, Alignment, and Engagement - Students' Prospects and Change During Their First Year in Computing Education." Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research, November 2020, pp. 1-5, article no. 34, doi: 10.1145/3428029.3428049.

Marwan, Samiha, et al. "Adaptive Immediate Feedback Can Improve Novice Programming Engagement and Intention to Persist in Computer Science." ICER '20: Proceedings of the 2020 ACM Conference on International Computing Education Research, August 2020, pp. 194-203, doi: 10.1145/3372782.3406264.

O'Leary, Ciaran, et al. "Developing a Software Engineering Curriculum for the Emerging Software Industry in China." 19th Conference on Software Engineering Education & Training (CSEET'06), IEEE, 19-21 April 2006, doi: 10.1109/CSEET.2006.16.

Petersen, Andrew, et al. "Revisiting why students drop CS1." Koli Calling '16: Proceedings of the 16th Koli Calling International Conference on Computing Education Research, 24 November 2016, pp. 71-80, doi: 10.1145/2999541.2999552

Phillips, Andrew T., et al. "Implementing CC2001: a breadth-first introductory course for a just-in-time curriculum design." ACM SIGCSE Bulletin, vol. 35, no. 1, Jan. 2003, pp. 238-242, doi: 10.1145/792548.611978.

Säde, Merilin, et al. "Factors That Influence Students' Motivation and Perception of Studying Computer Science." Proceedings of the 50th ACM Technical Symposium on Computer Science Education, February 2019, pp. 873-878, doi: 10.1145/3287324.3287395.

Säde, Merilin and Reelika Suviste. "Using a Programming MOOC as an Admission Mechanism for CS." Proceedings of the 2020 IEEE 20th International Conference on Advanced Learning Technologies (ICALT), 2020, pp. 42-44, doi:10.1109/ICALT49669.2020.00019.

Salguero, Adrian, et al. "Understanding Sources of Student Struggle in Early Computer Science Courses." ICER 2021: Proceedings of the 17th ACM Conference on International Computing Education Research, August 2021, pp. 319-333, doi: 10.1145/3446871.3469755.

Sharmin, Sadia. "Creativity in CS1: A Literature Review." ACM Transactions on Computing Education, vol. 22, no. 2, article 16, Nov. 2021, pp. 1-26. doi:10.1145/3459995.

Shields, Christopher. "Aristotle." The Stanford Encyclopedia of Philosophy, edited by Edward N. Zalta, Spring 2022 ed., 16 Mar. 2022, https://plato.stanford.edu/archives/spr2022/entries/aristotle/.

Stasiuk, Yurii. "Computer science expands hiring, course offerings to meet record high demand." *Yale Daily News*, 11 Sept. 2022, https://yaledailynews.com/blog/2022/09/11/computer-science-expands-hiring-course-offerings-to-meet-record-high-demand/.

Vega, Carlos, et al. "A scalable and incremental project-based learning approach for CS1/CS2 courses." Education and Information Technologies, vol. 18, no. 2, 2013, pp. 309-329, doi: 10.1007/s10639-012-9242-8.

Veilleux, Nanette, et al. "The Relationship Between Belonging and Ability in Computer Science." Proceedings of the 44th ACM Technical Symposium on Computer Science Education, March 2013, pp. 65–70, doi:10.1145/2445196.2445220.

Yale University Department of Computer Science. "Graduation Celebration." 24 May 2021, https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjszZ_12tr-AhUrFVkFHWp8AxcQFnoECBAQAQ&url=https%3A%2F%2Fcpsc.yale.edu%2Fsites%2Fdefault%2Ffiles%2Ffiles%2F2021%2520Commencement.pdf&usg=AOvVaw0vg-Fm-875W4TtnXfaEZYQ

Ye, Alex and Pranava Dhar. "Yale College offers new computing and linguistics joint major." Yale Daily News, 1 Nov. 2022, https://yaledailynews.com/blog/2022/11/01/yale-college-offers-new-computing-and-linguistics-joint-major/.

Notes

Two aspects of my proposal are not mentioned explicitly in this final report: an analysis of computer science job listings and a report on the computer science curriculum pathways of other top university CS programs. The rationale for the job listings analysis was that the course would focus on the languages and technologies demanded by industry. However, two considerations led me to pivot from this analysis. First, the resounding conclusion of the CSE literature and of industry surveys (included above) is that there is no consensus set of technologies that is most in-demand. Second, multiple studies indicate that, for the purposes of introductory curriculum, technologies are best chosen for their pedagogical appropriateness rather than their industry popularity. For anyone interested in in-demand industry technologies, the paper *Analysis of Software Engineering Industry Needs and Trends: Implications for Education* incorporates some great analysis of job listings data, and the StackOverflow developer survey is consistently insightful. See the latest version here: https://survey.stackoverflow.co/2022/

I did conduct the survey of other university computer science pathways and found for virtually all programs either a) the general university requirements exhibited so much influence over the major curriculum that it was not comparable to Yale's segmentation of major and distributional requirements or b) the university does substantially the same thing as Yale, having students take Data Structures, Algorithms, and Systems and filling the rest of their major coursework with electives.